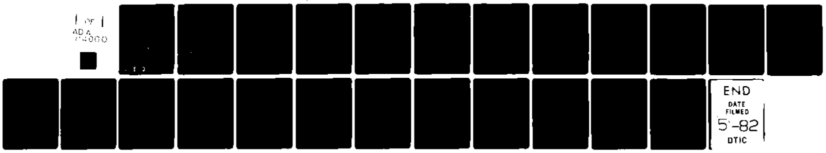
AD-A114 000 MARYLAND UNIV COLLEGE PARK DEPT OF COMPUTER SCIENCE F/G 9/2
A NEW VERIFICATION STRATEGY FOR ITERATIVE PROGRAMS.(U)
OCT 81 D D DUNLOP, V R BASILI F49620-80-C-0001
UNCLASSIFIED TR-1114 AFOSR-TR-82-0291 NL

1 OF 1
AD A
114000



END
DATE
FILMED
5-82
DTIC

(12)

AFOSR-TR- 82-0291

AD A114000

UNIVERSITY OF MARYLAND

COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

82 04 26 125

Technical Report TR-1114   October, 1981
[redacted]-F49620-80-C-0001

A New Verification Strategy For Iterative Programs*

Douglas D. Dunlop and Victor R. Basili

Department of Computer Science
University of Maryland
College Park, MD 20742

---

*This work was supported in part by the Air Force Office of Scientific Research Contract [redacted]-F49620-80-C-0001 to the University of Maryland. The material contained in this paper will become part of a dissertation to be submitted to the Graduate School, University of Maryland, by Douglas D. Dunlop, in partial fulfillment of the requirements for the Ph.D. degree in Computer Science.

ABSTRACT

A new verification strategy for iterative programs is described. The technique is based on the idea of applying a correctness/incorrectness preserving transformation to the program to be verified. The transformation is performed in such a way that the new program is substantially easier to verify than the original. Examples which illustrate the use of the technique are presented. The method is compared and contrasted with subgoal induction and the inductive assertion technique.

KEYWORDS and PHRASES: program verification, proof transformation, reduction hypothesis, subgoal induction, inductive assertion technique

CR CATEGORIES: 5.24

1. Introduction

The difficulties associated with the general verification of computer programs are well known. Chief among these difficulties is the problem of creating a suitable inductive hypothesis (e.g. loop invariant) for programs which use iteration or recursion. A large number of guidelines or heuristics for the construction of these hypotheses have appeared in the literature. Particularly promising are results in [Mills 75, Basu & Misra 75, Wegbreit 77, Morris & Wegbreit 77] which show that for a particular class of program/specification pairs, an adequate inductive hypothesis can be obtained in a straightforward deterministic manner. A consequence of this determinism is that for a verification problem in this class, the program can be proven/disproven correct with respect to its specification (assuming termination) by testing several verification conditions based on the specification and characteristics of the program components. Unfortunately, this class is rather restricted and, as a result, many verification problems which arise naturally in practice are not covered by these results.

In this report, a verification strategy is described which is based on the idea of applying a correctness/incorrectness preserving transformation to the program under consideration. The motivation behind the transformation is to produce a verification problem such that, in a manner similar to that in the above mentioned work, an inductive hypothesis can be created in a deterministic fashion. Thus we are proposing replacing the problem of synthesizing an inductive hypothesis with one of discovering an adequate correctness/incorrectness preserving transformation. In a number of examples we have studied, the latter problem appears to be more tractable.

In the remainder of this section we discuss a general verification problem which occurs often in practice and then suggest the idea of a transformation as a means to solve the problem. A number of heuristics for discovering an appropriate transformation are given in Section 2 and are illustrated with examples. In Section 3, the proposed solution is compared and contrasted with the inductive assertion and subgoal induction approaches to the verification problem. Finally, the application of the technique to more complex program forms is discussed in Section 4.

In our analysis, we will consider a program P of the form

```
X := K(X);
while B(X) do
    X := H(X)
    od;
X := Q(X)
```

where X is the program data state, K, H and Q represent data-state-to-data-state functions and B is a predicate on the data state. For the present we assume each of the functions and the predicate is explicitly known; this requirement regarding the function Q is relaxed in Section 4. We assume that the program specification is formulated as a function mapping the input data state to the portion of the output data state which corresponds to the program variables whose final values are of interest. We will use a function e which extracts from any data state the data state portion corresponding to these variables. If f is the specification function, the domain of f, written D(f), is the set of input data states for which a corresponding output is specified. Furthermore, the notation [P] will be used for the data-state-to-data-state function computed by the program P. Thus for this verification problem, P is correct with respect to its specification function f if and only if for any X in D(f), [P](X) is defined and f(X)=e([P](X)).

Let TAIL be the portion of P which follows the initialization X := K(X), i.e. TAIL is the WHILE loop followed by the assignment X := Q(X). We begin with the following:

<u>Observation 1</u> - P is correct with respect to (wrt) f if and only if (iff) TAIL is correct wrt the input/output relation

$$g = \{(X,Y) \mid \exists\, X0 \in D(f) \ni (K(X0)=X \;\&\; f(X0)=Y)\}.$$

That is, P is correct wrt f iff on data states X which are output by K on input X0 ∈ D(f), TAIL produces f(X0). For the moment, we make the following two assumptions:

FUNCTION: The relation g is a function, i.e.
$$K(X0)=K(X1) \rightarrow f(X0)=f(X1)$$
for all X0, X1 ∈ D(f), and

CLOSURE: The WHILE loop is closed [Basu & Misra 75] for the domain of g, i.e.
$$K(X0)=X,\ B(X) \rightarrow \exists\, X1 \in D(f) \ni K(X1)=H(X)$$
for all X0 ∈ D(f).

Given these two assumptions, it is easy to state the necessary and sufficient verification conditions for the correctness of TAIL (assuming termination) wrt g. These are (adapted from [Mills 75, Basu & Misra 75, Morris & Wegbreit 77])

$$X \in D(g),\ B(X) \rightarrow g(X)=g(H(X))$$
$$X \in D(g),\ \sim B(X) \rightarrow g(X)=e(Q(X))$$

and are called the <u>iteration</u> and <u>boundary</u> conditions, respectively, in [Misra 78]. They are equivalent to

$$X0 \in D(f),\ K(X0)=X,\ B(X) \rightarrow g(X)=g(H(X))$$
$$X0 \in D(f),\ K(X0)=X,\ \sim B(X) \rightarrow g(X)=e(Q(X))$$

i.e.

$$X0 \in D(f),\ B(K(X0)) \rightarrow g(K(X0))=g(H(K(X0)))$$
$$X0 \in D(f),\ \sim B(K(X0)) \rightarrow g(K(X0))=e(Q(K(X0)))$$

i.e.

$$X0,X1 \in D(f),\ B(K(X0)),\ K(X1)=H(K(X0)) \rightarrow g(K(X0))=g(K(X1))$$
$$X0 \in D(f),\ \sim B(K(X0)) \rightarrow g(K(X0))=e(Q(K(X0)))$$

i.e.

ITERATION:

```
  X0,X1 ∈ D(f), B(K(X0)), K(X1)=H(K(X0)) -> f(X0)=f(X1)
 BOUNDARY:
  X0 ∈ D(f)    ,~B(K(X0))                  -> f(X0)=e(Q(K(X0))).
```

We conclude as follows:

<u>Observation 2</u> - Suppose FUNCTION and CLOSURE hold. P is correct wrt f (assuming termination) iff ITERATION and BOUNDARY hold.

<u>Example 1</u> - Consider the following program

```
{a>=0}
a := a/2;
while a ∉ {0,1} do
   a := a - 2
   od;
a := (if a=0 then 1 else 0)
{a=EVEN(a0/2)}.
```

In this program, the data state consists of the single integer variable a and will be represented by <a>. The function EVEN(a) appearing in the program postcondition returns 1 if its argument is even and 0 otherwise. The specification function f for this program is

```
      f(<a0>)=a <-> a0>=0 & a=EVEN(a0/2)
```

which implies

```
      <a0> ∈ D(f) <-> a0>=0.
```

This program relates to the general program form above as follows

```
      K(<a0>)=<a> <-> a=a0/2
      B(<a>)      <-> a ∉ {0,1}
      H(<a0>)=<a> <-> a=a0-2
      Q(<a0>)=<a> <-> (a0=0 & a=1) OR (a0≠0 & a=0)
      e(<a>)=a.
```

Note that assumptions FUNCTION and CLOSURE hold. If the program terminates for all inputs in D(f), it is correct wrt its specification iff conditions ITERATION and BOUNDARY hold. These can be written

```
  a0>=0, a1>=0, a0/2 ∉ {0,1}, a1/2=a0/2 - 2
                                          -> EVEN(a0/2)=EVEN(a1/2)
```

and

```
  a0>=0, a0/2 ∈ {0,1} -> EVEN(a0/2)=(if a0/2=0 then 1 else 0)
```

respectively.

At this point we stop to consider the assumptions we have made. Suppose assumption FUNCTION is false, but CLOSURE holds. Thus we have X0, X1 ∈ D(f) satisfying

```
    K(X0)=K(X1) & f(X0)≠f(X1).
```

Now if B(K(X0)), then K(X2)=H(K(X0)) for some X2 ∈ D(f) by CLOSURE. If ITERATION was valid, we would have f(X0)=f(X2) as well as f(X1)=f(X2). Since this contradicts our hypothesis, ITERATION does not hold. On the other hand, suppose ~B(K(X0)). If BOUNDARY held, we would have f(X0)=e(Q(K(X0))) as well as f(X1)=e(Q(K(X1))). The contradiction leads us to conclude that if FUNCTION is false, but CLOSURE holds, (at least) one of ITERATION and BOUNDARY is false. Since FUNCTION being false implies P is not correct wrt f (i.e. [P](X0)=[P](X1), but f(X0)≠f(X1)), this leads us to the following:

Observation 3 - Suppose CLOSURE holds. P is correct wrt f (assuming termination) iff ITERATION and BOUNDARY hold.

Unfortunately, it is much more difficult to deal with the reliance on assumption CLOSURE. We can, however, gain insight into the problem by studying CLOSURE and its relation to condition ITERATION. Imagine P executing on some input in D(f) and consider the sequence of intermediate data states on which the predicate B is evaluated. CLOSURE requires that each of these intermediate states be "reachable" through the function K for some input element of D(f). It is this reachability that enables condition ITERATION to test whether this sequence of states stays on the right "track," or more specifically, that the inverse images of these states through K remain in the same level set of f.

Most often in practice, however, the purpose of the initialization K is to "constrain" the data state, thereby providing some specific "starting point" for the loop execution. In this case, the intermediate data states will not be reachable through K and, as a result, condition ITERATION may well hold for the simple reason that it is vacuously true, i.e. the term K(X1)=H(K(X0)) will be false for all X1. Rather than abandoning this approach to the verification of P, however, the solution suggested here is to "correct" the problem and proceed.

Consider substituting for K a suitable replacement initialization K′. By "suitable" here, we mean that the output of K′ must be sufficiently unconstrained so that CLOSURE holds for K′, and furthermore, that the substitution preserves the termination and correctness/incorrectness properties of the program P. The following definition formalizes this idea.

Definition - Let P′ be the program P with the initialization K replaced by K′. K′ is a reduction hypothesis iff CLOSURE holds for K′ and each of
  TERMINATES: For inputs in D(f), P terminates -> P′ terminates, and
  PRESERVES: P is correct wrt f iff P′ is correct wrt f
is satisfied.

The significance of the definition is that once a reduction hypothesis K′ has been located, we can prove/disprove the correctness of the original program by proving its termination and verifying ITERATION and BOUNDARY with K′ substituted for K.

The proposed solution of finding a reduction hypothesis is analogous to finding an adequate loop invariant for an inductive assertion proof [Hoare 69], or finding an appropriate loop function for a functional [Mills 75] or subgoal induction [Morris & Wegbreit 77] proof. We justify proposing an alternative to these standard techniques for the following reasons:

a) a reduction hypothesis has a unique intuition behind it; in a number of cases, this leads rather naturally to a solution,
b) the class of reduction hypotheses bears an interesting relationship to the class of adequate loop invariants for P,
c) unlike the technique of inductive assertions, it is possible to disprove an incorrect program without considering the program beyond the loop, i.e. by disproving ITERATION,
d) in the case there is no initialization (i.e. K is the identity function) and the loop is closed for D(f) (or, more generally, any time CLOSURE holds for K), a very efficient proof results since K itself is a reduction hypothesis, and
e) the reduction hypothesis solution provides continuity between the cases where CLOSURE does and does not hold; an understanding of "why" CLOSURE does not hold, for example, can provide insight into how to create K´ from K.

2. Constructing a Reduction Hypothesis

In this section we will consider several heuristics for creating a reduction hypothesis for P and will illustrate their use on example programs. We begin with some preliminary remarks.

In the discussion in the preceding section, we assumed that the program pieces corresponding to K, H and Q were deterministic, that is, we assumed their semantics (i.e. their input/output behavior) could be represented by data-state-to-data-state functions. The above results, however, extend in the natural way to the case where subprograms in P are nondeterministic provided one switches to a slightly more awkward relational notation for representing these subprograms. In particular, we will be interested in the case where the initialization is nondeterministic since it turns out that this kind of initialization is often a reasonable choice for a reduction hypothesis. If this new initialization is represented by a data-state-to-data-state relation K´, ITERATION and BOUNDARY translate to

ITERATION´:

$$X0,X1 \in D(f),\ K'(X0,X),\ B(X),\ K'(X1,H(X)) \rightarrow f(X0)=f(X1)$$

BOUNDARY´:

$$X0 \in D(f),\ K'(X0,X),\ \sim B(X) \rightarrow f(X0)=e(Q(X)),$$

where K´(X0,X) is a notation we will use which stands for (X0,X) $\in$ K´.

Now suppose this relation K´ satisfies CLOSURE, i.e.

$$K'(X0,X)\ \&\ B(X) \rightarrow \exists\ X1 \in D(f) \ni K'(X1,H(X))$$

for all X0 $\in$ D(f). The program P´ derived from P by replacing the initialization K with K´, i.e. by replacing

```
                X := K(X)
with
                X := "any Y satisfying K´(X,Y)",
```
is correct wrt f (assuming termination) iff ITERATION´ and BOUNDARY´ hold. If K´ has been chosen so as to be a reduction hypothesis, the original P is correct wrt f iff P terminates for inputs in D(f) and ITERATION´ and BOUNDARY´ hold.

As an aid in expressing nondeterministic program segments, we will use a notation defined in [Dijkstra 76]. Specifically, an execution of the program

```
     if B1(X) -> P1
      ⊤ B2(X) -> P2
       .
       .
       .
      | Bn(X) -> Pn
     fi
```
calls for the execution of <u>any</u> single program Pi provided guard Bi(X) holds.

We now consider two opposing alternatives to constructing K´. Each is based on a different philosophy for insuring that PRESERVES is satisfied. The first is a <u>program-oriented</u> approach and is characterized by selecting K´ so that the programs P and P´ are equivalent, i.e. so that P and P´ exhibit identical input/output behavior. Such a K´ trivially guarantees PRESERVES is satisfied. The other approach is a <u>specification-oriented</u> approach and is characterized by selecting K´ as a superset of K in such a way that executions of P´ which use the extended aspect of the initialization are guaranteed to be correct, i.e. execute in accordance with the program specification. Such a K´ satisfies PRESERVES since the correctness of P´ implies the correctness of P (since K´ is a superset of K) and the correctness of P implies the correctness of P´ (the additional execution paths in P´ are known to be correct). Thus each approach chooses a different technique aimed at meeting PRESERVES. If, in addition, CLOSURE holds for this new initialization and TERMINATES is satisfied, then K´ is a reduction hypothesis.

We will begin by considering several program-oriented heuristics and will make use of the following definition.

<u>Definition</u> - Let G be a data-state-to-data-state relation and let P´ be the program P with initialization K replaced by G. G is an <u>alternative initialization</u> to K iff P and P´ have identical input/output behavior for inputs in both D(f) and D(G).

Thus if G is an alternative initialization to K, then, for a restricted set of inputs, G can be used in place of K without affecting the externally observable behavior of P. In what follows, we will make use of the following properties which are derived trivially from this definition:

- K is an alternative initialization to itself,
- the union of any number of alternative initializations is an alternative initialization, and
- any subset of an alternative initialization is an alternative initialization.

Our interest in alternative initializations is due to the following theorem.

<u>Theorem 1</u> - Any alternative initialization G whose domain includes $\overline{D(f)}$ and which satisfies CLOSURE is a reduction hypothesis.

<u>Proof</u> - Let G meet the conditions stated in the theorem and let P' stand for the program P with initialization G substituted for K. From the definition of an alternative initialization, P and P' have identical input/output behavior over D(f). Thus TERMINATES and PRESERVES are trivially satisfied and G is a reduction hypothesis.

The implication of Theorem 1 is that a reduction hypothesis may be created by discovering a suitable alternative initialization. This is the basis for all program-oriented approaches to synthesizing a reduction hypothesis. The following theorem suggests three techniques for constructing alternative initializations.

<u>Theorem 2</u> (Program-Oriented Heuristics) - Let K1, K2 and K3 be any functions satisfying

```
    LONGCUT:  K1(X)=Y -> B(Y) & H(Y)=K(X)
   SHORTCUT:  K2(X)=Y -> B(K(X)) & Y=H(K(X))
      NOCUT:  K3(X)=Y -> B(K(X)) & B(Y) & H(Y)=H(K(X))
```

for all X ∈ D(f). Then each of K1, K2 and K3 is an alternative initialization.

<u>Proof</u> - We will make repeated use of the WHILE loop property

```
         B(X) -> [TAIL](X)=[TAIL](H(X)).
```

Let X ∈ D(f) and X ∈ D(K1). Then

```
  [TAIL](K1(X))=[TAIL](H(K1(X)))=[TAIL](K(X))=[P](X),
```

hence K1 is an alternative initialization. Let X ∈ D(f) and X ∈ D(K2). Then

```
  [TAIL](K2(X))=[TAIL](H(K(X)))=[TAIL](K(X))=[P](X),
```

hence K2 is an alternative initialization. Finally, let X ∈ D(f) and X ∈ D(K3). Then

```
  [TAIL](K3(X))=[TAIL](H(K3(X)))=[TAIL](H(K(X)))
                                =[TAIL](K(X))=[P](X),
```

hence K3 is also an alternative initialization.

The labels on the conditions in Theorem 2 are motivated by the effect replacing K with the alternative initializations has on the number of iterations of the WHILE loop. K1 causes the loop to execute 1 additional iteration, K2 saves the loop an iteration, and K3 has no effect on the number of iterations. The significance of the theorem is that any combination of K, K1, K2 and K3 whose domain includes D(f) and which satisfies CLOSURE is a reduction hypothesis.

We now define a specification-oriented heuristic for creating a reduction hypothesis.

<u>Theorem 3</u> (Specification-Oriented Heuristic) - Let G be any alternative initialization to K whose domain includes D(f) and let K4 be any function satisfying

VERYSHORTCUT: $K4(X)=Y \rightarrow \neg B(Y) \;\&\; e(Q(Y))=f(X)$.

Let K′ be the union of G and K4 and suppose K′ satisfies CLOSURE. Then K′ is a reduction hypothesis.

<u>Proof</u> - Let K′ be as stated in the theorem and let P′ stand for the program P with initialization K′ substituted for K. Let $X \in D(f)$ and suppose P terminates for input X. If P′ executes on X and the aspect of the initialization K′ from G is used, P′ must also terminate since G is an alternative initialization to K; if the aspect of the initialization K′ from K4 is used, P′ must terminate since the output of K4 causes B to evaluate to false. Hence in any case, P′ terminates on the input X and TERMINATES holds. The remainder of the proof consists of showing that PRESERVES is satisfied. Suppose P is correct wrt f. Let $X \in D(f)$. If P′ executes on X and the aspect of the initialization K′ from G is used, then P′ is also correct on the input X since G is an alternative initialization to K; if the aspect of the initialization K′ from K4 is used, then P′ must produce

$e([TAIL](K4(X)))=e(Q(K4(X)))=f(X)$,

hence P′ is again correct on the input X. Thus in any case, the program P′ is correct wrt f. To show the converse, suppose P′ is correct wrt f. We must show that this implies P is correct wrt f. Let $X \in D(f)$. Then $X \in D(G)$. By the correctness of P′ and the fact that K′ includes G, the program P′′ derived from P by replacing the initialization K with G produces f(X) for the input X. Since G is an alternative initialization to K, P and P′′ exhibit identical input/output behavior for X, hence P produces f(X) for the input X and thus P is correct wrt f. This completes the proof of the theorem.

As in Theorem 2, the label on the condition in Theorem 3 is used to suggest the effect the initialization K4 has on the execution of the program. The output of a function satisfying VERYSHORTCUT causes the predicate B to evaluate to false, and consequently, the WHILE loop in P will execute zero times. We repeat that the functions K1, K2 and K3 represent program-oriented heuristics since they are designed specifically to preserve the effect of the program P (possibly over a restricted domain). Function K4, on the other hand, represents a specification-oriented heuristic since its purpose is to insure program behavior which is in agreement with the program specification. Now that these heuristics have been defined, we will illustrate their use with a number of examples.

Example 2 - This example illustrates a circumstance in which assumption CLOSURE "almost" holds but "not quite." In this situation it may be clear how to "expand" the initialization in order to satisfy CLOSURE in such a way that preserves the effect of the program. The program

```
{a>=0}
a := a*2;
while a>0 do
   a := a - 1;
   b := b + 1
   od;
a := b
{a=a0*2 + b0}
```

does not satisfy CLOSURE since only even values of a are output from the initialization K and the WHILE loop is not closed for this set. This problem could be "fixed" by supplementing K with another initialization, selected nondeterministically, which produced odd values of a. The combined effect would be an initialization which was capable of producing all (nonegative) values of a and would consequently satisfy CLOSURE. Based on this reasoning, we could supplement K with the initialization

```
    a := a*2 + 1,
```

but an inspection of the loop-body text indicates that we must compensate in this case by subtracting 1 from b if the effect of the program is to be preserved. This is actually an application of heuristic LONGCUT of Theorem 2; specifically, the initialization

```
   K1(<a0,b0>)=<a,b> <-> a=a0*2+1 & b=b0-1.
```

satisfies LONGCUT and is thus (by Theorem 2) an alternative initialization. In the above notation for nondeterminacy, the combined initialization may be written

```
     if TRUE -> a := a*2
     ▯  TRUE -> a := a*2 + 1; b := b-1
     fi.
```

and the relation K´ which represents this initialization is

```
 K´(<a0,b0>,<a,b>) <-> ((a=a0*2 & b=b0) OR (a=a0*2+1 & b=b0-1)).
```

Since K´ is the union of two alternative initializations (K and K1), K´ itself is an alternative initialization, and thus, applying Theorem 1, it is a reduction hypothesis. Hence the program can be verified correct with respect to its specification (assuming termination) by showing that ITERATION´ and BOUNDARY´ hold. Condition ITERATION´ has the following two aspects, which correspond to the cases where a is even and odd, respectively:

```
  a0>=0, a1>=0, a=a0*2, b=b0, a>0, a1*2+1=a-1, b1-1=b+1
                                              -> a0*2+b0=a1*2+b1
```

and

```
  a0>=0, a1>=0, a=a0*2+1, b=b0-1, a>0, a1*2=a-1, b1=b+1
                                              -> a0*2+b0=a1*2+b1.
```

These can be simplified (by eliminating a0, b0, a1 and b1) to

```
   a>0 -> a+b=(a-2)+(b+2)
```

and

```
    a>0 -> (a-1)+(b+1)=(a-1)+(b+1),
```

both of which hold. Condition BOUNDARY´ is

```
    a0>=0, a=a0*2, b=b0, a<=0 -> a0*2+b0=b
```

which simplifies to

```
    a=0 -> a+b=b,
```

and thus also holds.

<u>Example 3</u> - We noted above that the purpose of loop initialization is often to set the data state to some specific "starting point" for the execution of the loop. Once the loop begins execution, the data state leaves this starting point and takes on more general values. With this in mind, we note that the program P is equivalent to the program

```
X := H(K(X));
while B(X) do
   X := H(X)
   od;
X := Q(X)
```

on executions of P which require at least one loop iteration. Since the initialization in this new program includes an execution of the loop body, the hope is that the output of the new initialization will be general enough to satisfy CLOSURE. This observation is the motivation behind heuristic SHORTCUT in Theorem 2 and can be applied as follows: convert (if necessary) the verification problem to one over a domain where the loop will execute at least once and try the initialization H o K for the reduction hypothesis K´ (where "o" stands for function composition). As an illustration, consider the program

```
pa := 0;
while s ≠ NULL do
   pa := pa + head(s);
   s := tail(s)
   od;
if pa <= 0 then pa := 0 else pa := 1 fi
{pa=PAVG(s0)}
```

which determines whether the arithmetic average of the integers appearing in the sequence s is positive or negative. The function head(s) for a nonempty sequence s returns the lead element in s, tail(s) returns s with head(s) removed, and NULL denotes the sequence containing 0 elements. The function PAVG(s0) appearing in the postcondition has the value 1 if the average of the elements of s0 is positive and 0 otherwise. CLOSURE is not satisfied for this program since pa takes on values other than 0 as the loop iterates. Suppose we can convince ourselves that the program executes correctly when the input s is the empty sequence and the loop is bypassed. The remainder of the proof then consists of verifying the program assuming the precondition {s≠NULL}. Accordingly, we will now use the program specification function

f(<pa0,s0>)=pa <-> s0≠NULL & pa=PAVG(s0).
The heuristic suggested above is to try the initialization
K2(<pa0,s0>,<pa,s>) <-> s0≠NULL & pa=head(s0) & s=tail(s0)
as a reduction hypothesis. This function satisfies SHORTCUT and is thus an alternative initialization. Its domain includes D(f) (i.e. {<pa,s> | s≠NULL}) and since it satisfies CLOSURE, it is a reduction hypothesis. To apply this result in a demonstration of the correctness of the program, we must show ITERATION´ and BOUNDARY´ hold (using K2 for K´). Condition ITERATION´ is
s0≠NULL, s1≠NULL, pa=head(s0), s=tail(s0), s≠NULL,
pa+head(s)=head(s1), tail(s)=tail(s1) -> PAVG(s0)=PAVG(s1)
which simplifies to
s≠NULL -> PAVG(<pa>||s)=PAVG(<pa+head(s)>||tail(s)).
where <x> is the sequence containing the single element x and || denotes concatenation of sequences. Condition BOUNDARY´ is
s0≠NULL, pa=head(s0), s=tail(s0), s=NULL
-> PAVG(s0)=POSITIVE(pa)
where POSITIVE(pa) is a function which has the value 1 if pa is positive and 0 otherwise. This condition simplifies to
s0=<pa> -> PAVG(s0)=POSITIVE(pa).

Example 4 - We now illustrate an application of the specification-oriented heuristic VERYSHORTCUT of Theorem 3. Since any output Y of a function K4 satisfying VERYSHORTCUT must satisfy ~B(Y), this approach is most appropriately used when CLOSURE holds for all loop iterations except the last, i.e. the only lack of values in the range of K is where B is FALSE. The idea is to add to the initialization the production of these values in such a way that assures the correct execution of the program. The following program serves to illustrate this kind of technique. It searches a linked list for an element containing a key field with the value 17:

```
found := FALSE;
while p ≠ NIL & ~found do
  if p^.key=17 then
     found := TRUE
  else p := p^.link fi
  od
{found=INCHAIN(p0,17)}
```

The program notations p^.key and p^.link are the key and link fields, respectively, of the node pointed to by p. A link field of NIL is used to mark the end of the list. The function INCHAIN(p0,17) appearing in the postcondition is a predicate which holds iff the chain pointed to by p0 contains an element with a key field of 17. Note that since there is no program text following the loop, Q is the identity function. CLOSURE is not satisfied for this program since the variable found may take on the value TRUE on the last loop iteration. The initialization is extended by alternatively assigning the value TRUE to found, but only in the case where INCHAIN(p,17) holds. Specifically, we propose to supplement the given initialization with the function

```
  K4(<found0,p0>)=<found,p> <-> INCHAIN(p0,17) & p=p0 & found
```
to yield the new initialization
```
    if TRUE -> found := FALSE
     | INCHAIN(p,17) -> found := TRUE
    fi.
```
Note that K4 satisfies VERYSHORTCUT, i.e. on executions of the program which follow the second path, the loop body is bypassed and the program necessarily behaves in accordance with its specification. The combined initialization is represented by the input/output relation
```
  K'(<found0,p0>,<found,p>) <->
                       (~found OR found=INCHAIN(p0,17)) & p=p0.
```
Since arbitrary values of found and p may emerge from this initialization, CLOSURE is satisfied. By Theorem 3 (using the original initialization K for G), K' is a reduction hypothesis. We now use K' to state the necessary and sufficient conditions (assuming termination) for the correctness of the program. Corresponding to the two paths through the loop body of this program, there are two ITERATION' conditions
```
  (~found OR found=INCHAIN(p0,17)), p=p0, p≠NIL, ~found,
     p^.key=17, (~TRUE OR TRUE=INCHAIN(p1,17)), p=p1
                                 -> INCHAIN(p0,17)=INCHAIN(p1,17)
```
and
```
  (~found OR found=INCHAIN(p0,17)), p=p0, p≠NIL, ~found,
     p^.key≠17, (~FALSE OR FALSE=INCHAIN(p1,17)), p^.link=p1
                                 -> INCHAIN(p0,17)=INCHAIN(p1,17),
```
which simplify to
```
  ~found, p≠NIL, p^.key=17, INCHAIN(p,17)
                                 -> INCHAIN(p,17)=INCHAIN(p,17)
```
and
```
  ~found, p≠NIL, p^.key≠17
                            -> INCHAIN(p,17)=INCHAIN(p^.link,17),
```
respectively. Condition BOUNDARY' is
```
  (~found OR found=INCHAIN(p0,17)), p=p0, (p=NIL OR found)
                                 -> found=INCHAIN(p0,17),
```
which simplifies to
```
  (found & INCHAIN(p,17)) OR (~found & p=NIL)
                                 -> found=INCHAIN(p,17).
```

Example 5 - This example illustrates a circumstance where several of the above heuristics are applied in order to create a reduction hypothesis. The following program sums the elements of a sequence:

```
        {s≠NULL}
        sum := 0;
        while s≠NULL do
           sum := sum + head(s);
           s := tail(s)
           od;
        {sum=SUM(s0)}.
```

The notation SUM(s0) appearing in the postcondition stands for the summation of the elements of s0. We begin our reasoning as follows. CLOSURE is not satisfied for this program because sum is constrained to the value 0 by an assignment statement. What would be the effect on the program of removing this assignment? After the first loop iteration, sum would have the value sum0+head(s0), rather than head(s0). We could compensate for this discrepancy by subtracting sum0 from head(s0) before the loop begins execution. Thus we choose to replace "sum:=0" with "head(s):=head(s)-sum." This is an application of heuristic NOCUT of Theorem 2. Indeed, the function corresponding to this new initialization,

```
  K3(<sum0,s0>)=<sum,s>) <-> s0≠NULL & s≠NULL & sum=sum0 &
                    head(s)=head(s0)-sum0 & tail(s)=tail(s0),
```

satisfies NOCUT and is thus an alternative initialization. We note that CLOSURE is satisfied for this initialization on all but the final loop iteration (i.e. K3 cannot produce an output which satisfies s=NULL). As in Example 4, the solution under these circumstances is to supplement the initialization with a function satisfying VERYSHORTCUT; in this example such a function would be

```
  K4(<sum0,s0>)=<sum,s>) <-> sum=SUM(s0) & s=NULL.
```

Note that the output of K4 causes B to evaluate to FALSE and forces the program to be correct wrt its specification. The complete initialization is then

```
      if s≠NULL -> head(s) := head(s) - sum
      [] TRUE  -> sum := SUM(s); s := NULL
      fi.
```

Let K′ represent this initialization (i.e. let K′ be the union of K3 and K4). By Theorem 3 (using K3 for G), K′ is a reduction hypothesis. Corresponding to whether K3 or K4 is used in the term K′(X1,H(X)), condition ITERATION′ has the following two aspects:

```
  s0≠NULL, sum=sum0, head(s)=head(s0)-sum0, tail(s)=tail(s0),
    s≠NULL, s1≠NULL, tail(s)≠NULL, sum+head(s)=sum1,
    head(tail(s))=head(s1)-sum1, tail(tail(s))=tail(s1)
                -> SUM(s0)=SUM(s1)
```

and

```
  s0≠NULL, sum=sum0, head(s)=head(s0)-sum0, tail(s)=tail(s0),
    s≠NULL, sum+head(s)=SUM(s1), tail(s)=NULL
                -> SUM(s0)=SUM(s1)
```

which simplify to

```
  s≠NULL, tail(s)≠NULL ->
               sum+head(s)=SUM(tail(s))=
               sum+head(s)+head(tail(s))+SUM(tail(tail(s)))
```

and

```
  tail(s)=NULL -> sum+head(s)+SUM(tail(s))=sum+head(s).
```

Condition BOUNDARY′ is

```
  sum=SUM(s0), s=NULL -> SUM(s0)=sum.
```

3. Relation to Standard Correctness Techniques

In this section, we discuss the relationship between the proposed verification strategy and the subgoal induction [Morris & Wegbreit 77] (see also [Manna & Pnueli 70, Manna 71]), and inductive assertion [Hoare 69] correctness techniques. We will define these methods in the framework of the verification problem described above, i.e. in each case we wish to prove/disprove the program P correct wrt its specification function f.

All three techniques call for creating and verifying an hypothesis concerning some aspect of the behavior of P and then applying the hypothesis to prove/disprove the correctness of the program. In the proposed technique, this hypothesis is a reduction hypothesis; in subgoal induction, we will refer to the hypothesis as a <u>tail function</u>; in the inductive assertion technique, the hypothesis is an adequate <u>inductive assertion</u>.

A tail function g in a subgoal induction proof is a general description of the input/output behavior of program TAIL. Specifically, g has the same functionality as the specification f and must satisfy each of

SI1: $X \in D(g)$, $B(X) \rightarrow H(X) \in D(g)$
SI2: $X \in D(g)$, $B(X) \rightarrow g(X)=g(H(X))$
SI3: $X \in D(g)$, $\sim B(X) \rightarrow g(X)=e(Q(X))$
SI4: $X \in D(g) \rightarrow$ TAIL terminates with input X
SI5: $X \in D(f) \rightarrow K(X) \in D(g)$.

The first four of these conditions establish that TAIL is correct wrt g, SI5 assures that D(g) is sufficient for testing the correctness of the program P. When such a function g has been found, the program P is correct wrt f iff

SI6: $X \in D(f) \rightarrow g(K(X))=f(X)$.

In the inductive assertion technique, an adequate inductive assertion is a sufficiently strong invariant relation between initial data states and data states occurring at the loop predicate B. Specifically, an adequate inductive assertion is a binary relation A over the data state of P which satisfies

IA1: $X \in D(f) \rightarrow A(X,K(X))$
IA2: $X \in D(f)$, $A(X,Y)$, $B(Y) \rightarrow A(X,H(Y))$
IA3: $X \in D(f)$, $A(X,Y) \rightarrow$ TAIL terminates with input Y
IA4: $X \in D(f)$, $A(X,Y)$, $\sim B(Y)$, $A(X,Z)$, $\sim B(Z)$
$\rightarrow e(Q(Y))=e(Q(Z))$.

If we view X and Y as representing the initial and current data states, IA1 and IA2 prove that A(X,Y) is a "loop invariant," IA3 is the necessary termination condition based on A, and IA4 tests whether A is sufficiently strong to verify the correctness of the program. When such a relation A has been found, the program P is correct wrt f iff

IA5: $X \in D(f)$, $A(X,Y)$, $\sim B(Y) \rightarrow f(X)=e(Q(Y))$.

The results of this section are contained in the following two theorems. They define a relationship between the three forms of program hypotheses and give a technique for transforming tail functions and adequate inductive assertions into reduction hypotheses.

Theorem 4 - Let g be a tail function for a subgoal induction proof of P, and let K′ be a relation defined by

K′(X0,X) <-> g(K(X0))=g(X).

Then K′ is a reduction hypothesis.

Proof - We must prove that K′ satisfies CLOSURE and that TERMINATES and PRESERVES hold where P′ is P with initialization K′ substituted for K. To see that K′ satisfies CLOSURE, let

X0 ∈ D(f) & K′(X0,X) & B(X)

i.e.

X0 ∈ D(f) & g(K(X0))=g(X) & B(X).

By SI1 and SI2, g(X)=g(H(X)); hence g(K(X0))=g(H(X)) and

X0 ∈ D(f) & K′(X0,H(X)),

thus CLOSURE holds for K′. TERMINATES must be satisfied since the range of K′ contains only elements of D(g) and SI4 is satisfied. Finally, to show that PRESERVES holds, we will prove that P and P′ compute the same function (in the variables of interest) over the domain D(f). Let X0 ∈ D(f). On input X0, P produces a result Y satisfying

K(X0)=X & Y=e([TAIL](X))

for some state X. On input X0, P′ produces a result Y′ satisfying

K′(X0,X′) & Y′=e([TAIL](X′))

for some state X′. These may be rewritten

K(X0)=X & Y=g(X)

and

g(K(X0))=g(X′) & Y′=g(X′)

using the definition of K′ and the fact that SI1-SI4 imply TAIL computes g. These imply

Y=g(K(X0))=Y′,

hence P and P′ compute the same function over D(f).

Thus a tail function g together with program initialization K can be used to construct a reduction hypothesis. The (perhaps) surprising result in the following theorem is that an adequate inductive assertion (by itself) is a reduction hypothesis, i.e. the class of reduction hypotheses for P contains the class of adequate inductive assertions for P.

Theorem 5 - If A is an adequate inductive assertion for P, then A is a reduction hypothesis for P.

Proof - We must show A satisfies CLOSURE and that TERMINATES and PRESERVES hold where P′ is P with initialization A substituted for K. To see that A satisfies CLOSURE, let

X0 ∈ D(f) & A(X0,X) & B(X).

By IA2, A(X0,H(X)), thus CLOSURE holds for A. TERMINATES follows directly from IA3. Finally, to show that PRESERVES holds, we will show that P and P′ compute the same function (in the variables of interest) over the domain D(f). Let X0 ∈ D(f). On input X0, P will produce Y satisfying

K(X0)=X & Y=e([TAIL](X))

for some state X. By IA1, A(X0,X). By IA3 the WHILE loop terminates on input X giving some result T where ~B(T). Repeated application of IA2 and the loop property

```
    B(Z) -> [TAIL](Z)=[TAIL](H(Z))
```

gives A(X0,T) and [TAIL](X)=[TAIL](T)=Q(T). Thus on input X0, P produces Y satisfying

```
  (1)  A(X0,T) & ~B(T) & Y=e(Q(T))
```

for some T. P´ on the other hand, with input X0 will produces Y´ satisfying

```
    A(X0,X´) & Y´=e([TAIL](X´))
```

for some state X´. Again by IA3, the WHILE loop terminates on input X´ giving some result T´ where ~B(T´). Repeated application of IA2 and the above loop property gives A(X0,T´) and [TAIL](X´)=[TAIL](T´)=Q(T´). Thus P´ produces Y´ satisfying

```
  (2)  A(X0,T´) & ~B(T´) & Y´=e(Q(T´))
```

for some T´. In light of IA4, (1) and (2) imply Y=Y´, thus P and P´ compute the same function over D(f).

<u>Example 6</u> - To illustrate these ideas, we consider again the program discussed in Example 2:

```
{a>=0}
a := a*2;
while a>0 do
   a := a-1;
   b := b+1
   od;
a := b
{a=a0*2 + b0}.
```

The following are possible tail functions for a subgoal induction proof of the program

```
    g1(<a0,b0>)=a <-> a0>=0 & a=a0+b0
    g2(<a0,b0>)=a <-> a=MAX(a0,0)+b0.
```

Several adequate inductive assertions are

```
  A1(<a0,b0>,<a,b>) <-> b=b0+2*a0-a & a>=0
  A2(<a0,b0>,<a,b>) <-> b=b0+2*a0-a & a>=0 & a0>=0
  A3(<a0,b0>,<a,b>) <-> b=b0+2*a0-a & a0>=a>=0
  A4(<a0,b0>,<a,b>) <-> b=b0+2*MAX(a0,0)-MAX(a,0).
```

By Theorem 4,

```
  K1(<a0,b0>,<a,b>) <-> g1(K(<a0,b0>))=g1(<a,b>)
```

and

```
  K2(<a0,b0>,<a,b>) <-> g2(K(<a0,b0>))=g2(<a,b>),
```

i.e.

```
  K1(<a0,b0>,<a,b>) <-> 2*a0>=0 & a>=0 & 2*a0+b0=a+b
```

and

```
  K2(<a0,b0>,<a,b>) <-> MAX(2*a0,0)+b0=MAX(a,0)+b
```

are reduction hypotheses. Theorem 5 states that each of A1-A4 is also a reduction hypothesis. Thus any of these can be used in place of K´ in the proof in Example 2. We remark that the relation K´ used in that example is not an inductive assertion, hence the class of adequate inductive assertions is a proper subset of the class of reduction hypotheses.

4. Proof Transformations

Applying the verification technique proposed above based on ITERATION′ and BOUNDARY′ requires ascertaining the input/output behavior of the loop initialization, loop body and program text following the loop. In many cases, this ascertaining process may be difficult (e.g. if these program segments contain additional WHILE loops). In view of this problem, we now consider the situation which occurs when the input/output behavior of the program text following the WHILE loop is not explicitly known. The verification problem under consideration, then, will be of the form

```
{X0 ∈ D(f) & X=X0}
X := K(X);
while B(X) do
   X := H(X)
   od;
T
{e(X)=f(X0)}
```

where K, B and H are as before and T represents some unspecified block of program text. Again our intention is to prove/disprove the program correct wrt its specification function f.

Suppose we have a reduction hypothesis K′ for P and can show condition ITERATION′ holds. What sense does BOUNDARY′ make and how can we proceed? In this situation, condition BOUNDARY′ corresponds to a new but simpler correctness problem. Specifically, we must prove

```
{X0 ∈ D(f) & K′(X0,X) & ~B(X)}
T
{e(X)=f(X0)}.
```

This problem is simpler than the original due to the fact that the loop has been eliminated from the program. Thus we have used a reduction hypothesis and condition ITERATION′ to transform the correctness question for the original program P to a correctness question for a substantially simpler program. If the program T contains further looping structures, the process may be repeated.

Example 7 - The following program operates on sequences a, x, y and z of natural numbers. The function head(s) of a nonempty sequence s is the leftmost element of s, and tail(s) is s with head(s) removed. The infix operator || denotes concatenation of sequences and NULL denotes the sequence with 0 elements. The predicate odd(a) is true iff a is odd.

```
while a ≠ NULL do
   if odd(head(a)) then y := y || <head(a)>
                   else z := z || <head(a)> fi;
   a := tail(a)
   od;
while y ≠ NULL do
   x := x || <head(y)>; y := tail(y)
   od;
while z ≠ NULL do
   x := x || <head(z)>; z := tail(z)
   od
{x=F(x0,y0,z0,a0)}
```

The function F appearing in the postcondition is defined as
  F(x,y,z,a) = x || y || ODDS(a) || z || EVENS(a)
where ODDS(a) is the sequence containing the odd elements of a in the order they appear in a (EVENS(a) is similar). In this example, T corresponds to the last two WHILE loops and since there is no initialization, K is the identity function. Since CLOSURE holds for this function, K itself is a reduction hypothesis. Corresponding to the two paths through the first loop body, there are the two ITERATION′ conditions, namely

```
  a≠NULL,  odd(head(a))
                      -> F(x,y,z,a)=F(x,y||<head(a)>,a,tail(a))
  a≠NULL, ~odd(head(a))
                      -> F(x,y,z,a)=F(x,y,z||<head(a)>,tail(a)).
```

Once these have been proven, the verification problem then transforms to

```
{a=NULL}
while y ≠ NULL do
   x := x || <head(y)>; y := tail(y)
   od;
while z ≠ NULL do
   x := x || <head(z)>; z := tail(z)
   od
{x=F(x0,y0,z0,a0)}.
```

Here T corresponds to the last loop and again K is the identity function, CLOSURE holds and hence K itself is a reduction hypothesis. The proof of this program thus consists of showing ITERATION′, i.e.

```
  a=NULL, y≠NULL -> F(x,y,z,a)=F(x||<head(y)>,tail(y),z,a)
```

and then verifying

```
{a=NULL & y=NULL}
while z ≠ NULL do
   x := x || <head(z)>; z := tail(z)
   od
{x=F(x0,y0,z0,a0)}.
```

Again using the identity function as a reduction hypothesis, this remaining verification problem can be proved by showing

ITERATION′ and BOUNDARY′, i.e.

$a=NULL,\ y=NULL,\ z \neq NULL \to$
$$F(x,y,z,a)=F(x \| \langle head(z) \rangle, y, tail(z), a)$$

and

$a=NULL,\ y=NULL,\ z=NULL \to x=F(x,y,z,a)$.

We remark that none of the program-oriented reduction hypothesis techniques defined in Theorem 2 assume any knowledge of the characteristics of T. Thus these techniques can be used for creating a reduction hypothesis in the circumstance where the input/output behavior of this subprogram is not known. The specification-oriented heuristic of Theorem 3, however, cannot be employed without this knowledge.

5. Concluding Remarks

From a practical point of view, it is difficult to carefully assess the relative merits of the proposed program verification methodology. Preferences by people involved in verifying programs are often based on which methodology appears to be more "natural" or "intuitive" in a given application. Furthermore, answers to questions such as these no doubt are largely influenced by the way a person was trained in the field of software engineering.

Despite this caveat, we offer our view that in a number of cases, it seems "easier" to create a reduction hypothesis than it does to create an adequate inductive assertion or tail function for a proof by the standard techniques discussed in Section 3. Indeed, it is difficult to argue the reverse case in light of the results of that section, which state that any adequate inductive assertion is a reduction hypothesis, and that any tail function can be simply transformed to one. On the other hand, our feeling is that the verification conditions which result from the use of a reduction hypothesis seem somewhat more complicated than their counterparts in either of these standard techniques. This is largely due to the necessity of having three distinct data states appearing in condition ITERATION′ and two distinct data states appearing in condition BOUNDARY′. Both of these verification conditions, however, are usually easily simplified in practice.

In light of these comments, an interesting direction for future research is the translation of heuristics such as those defined in Theorems 2 and 3 into the framework of standard correctness techniques. For example, what, if any, is the impact of these correctness/incorrectness preserving transformations on the synthesis of an inductive assertion or tail function for the program under consideration? Preliminary results along this line appear in [Dunlop & Basili 81]. In that report, heuristic SHORTCUT is applied to the problem of synthesizing tail functions for initialized loop programs and meets with a fair degree of success.

The solution of any complex problem is often best decomposed into solutions of appropriate simpler problems. This is an important principle on which our verification strategy is based. A search for a reduction hypothesis is really a search for a suitable simpler problem to substitute for the original. As discussed in Section 4, proving condition ITERATION' for this simpler problem decomposes the solution of the new problem into the solution of a still simpler problem and so on.

6. References

[Basu & Misra 75]
Basu, S. and Misra, J. Proving Loop Programs, IEEE Transactions on Software Engineering, Vol. SE-1, March 1975, pp. 76-86.

[Dijkstra 76]
Dijkstra, E. W. A Discipline of Programming, Prentice-Hall, 1976.

[Dunlop & Basili 81]
Dunlop, D. and Basili, V. A Heuristic For Deriving Loop Functions, University of Maryland Computer Science Technical Report TR-1115, October 1981.

[Hoare 69]
Hoare, C. A. R. An Axiomatic Basis for Computer Programming, CACM, Vol. 12, Oct. 1969, pp. 576-583.

[Manna 71]
Manna, Z. Mathematical Theory of Partial Correctness, J. Computer System Sci., Vol. 5, June 1971, pp. 239-253.

[Manna & Pnueli 70]
Manna, Z. and Pnueli, A. Formalization of Properties of Functional Programs, JACM, Vol. 17, July 1970, pp. 555-569.

[Mills 75]
Mills, H. D. The New Math of Computer Programming, CACM, Vol. 18, Jan. 1975, pp. 43-48.

[Misra 78]
Misra, J. Some Aspects of the Verification of Loop Computations, IEEE Transactions on Software Engineering, Vol. SE-4, Nov. 1978, pp. 478-486.

[Morris & Wegbreit 77]
Morris, J. H. and Wegbreit, B. Subgoal Induction, CACM, Vol. 20, April 1977, pp. 209-222.

[Wegbreit 77]
Wegbreit, B. Complexity of Synthesizing Inductive Assertions, JACM, Vol. 24, July 1977, pp. 504-512.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 82-0291	2. GOVT ACCESSION NO. AD-A114 800	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A NEW VERIFICATION STRATEGY FOR ITERATIVE PROGRAMS		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL
		6. PERFORMING ORG. REPORT NUMBER TR-1114
7. AUTHOR(s) Douglas D. Dunlop and Victor R. Basili		8. CONTRACT OR GRANT NUMBER(s) F49620-80-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science University of Maryland College Park MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Mathematical & Information Sciences Directorate Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE OCT 81
		13. NUMBER OF PAGES 22
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Program verification; proof transformation; reduction hypothesis; subgoal induction; inductive assertion technique.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A new verification strategy for iterative programs is described. The technique is based on the idea of applying a correctness/incorrectness preserving transformation to the program to be verified. The transformation is performed in such a way that the new program is substantially easier to verify than the original. Examples which illustrate the use of the technique are presented. The method is compared and contrasted with subgoal induction and the inductive assertion technique.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DATE
FILMED

-8